Maciej PETKO, Michał LUBIENIECKI, Michał STAWORKO
AKADEMIA GÓRNICZO-HUTNICZA, WYDZIAŁ INŻYNIERII MECHANICZNEJ I ROBOTYKI,
KATEDRA ROBOTYKI I MECHATRONIKI

Automatyczna implementacja programowo-sprzętowa algorytmów sterowania w układach FPGA

Dr hab. inż. Maciej PETKO

Adiunkt w Katedrze Robotyki i Mechatroniki AGH. Jego zainteresowania skupiają się na mechatronice, robotyce, zagadnieniach prototypowania i implementacji algorytmów przetwarzania sygnałów, głównie w sterowaniu i diagnostyce technicznej.

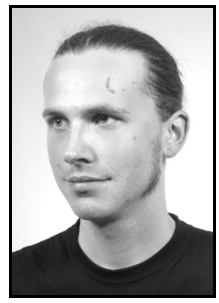
e-mail: petko@agh.edu.pl

Mgr inz. Michał LUBIENIECKI

Ukończyl studia na Wydziale Inzynierii Mechanicznej i Robotyki AGH (2007). Obecnie doktorant w Katedrze Robotyki i Mechatroniki AGH, współpracuje z Centrum Badawczym Fiata jako ESR (Early Stage Researcher) w ramach projektu „Smart Structures". Jako główne zainteresowania należy wymienić mechatronikę, teorię regulacji oraz zastosowanie materiałów piezoelektrycznych w tłumieniu drgań.

Mgr inż. Michał STAWORKO

Ukończył studia na Wydziale Inżynierii Mechanicznej i Robotyki AGH w 2007 roku. Obecnie jest doktorantem w Katedrze Robotyki i Mechatroniki AGH. Jego zainteresowania naukowe skupiają się na mechatronice, zagadnieniach implementacji algorytmów przetwarzania sygnałów w układach programowalnych i systemach wbudowanych.



Streszczenie

W artykule przedstawiono procedurę sprzętowo-programowej implementacji algorytmów sterowania w systemie w układach programowalnych opartej na automatycznej generacji kodu części sprzętowej i programowej ze schematu Simulinka. Opracowany generator kodu umożliwia syntezę komponentów sprzętowych, kompilację programu z interfejsami części sprzętowej dla mikroprocesora typu *soft-core* oraz dokładną symulację zaimplementowanego algorytmu w Simulinku. Metodologia i narzędzie zostały zweryfikowane na przykładzie sterownika robota równoległego.

Słowa kluczowe: implementacja sterowania, architektury sterowników, układy programowalne, FPGA.

Automatic software-hardware implementation of control algorithms in FPGA

Abstract

The paper presents a procedure of control algorithms hardware-software implementation in a System-on-a-Programmable-Chip based on automatic generation of a code for hardware and software parts and their interfaces from a Simulink model. The developed code generator allows for synthesis of hardware components, compilation of a program with hardware interfaces for the soft-processor and accurate simulation of the implemented algorithm in Simulink. The methodology and tools were verified in a case study of a parallel robot control algorithm. Section 2 contains assumptions for the procedure, Section 3 - its description. Section 4 covers the automation method and describes functions of the code generator software. The automated design flow that includes the code generator is shown in Fig. 2. The code generator products and their use are presented in Fig. 3. The example of the parallel robot controller implementation is given in Section 5. The robot control algorithm scheme in Simulink is presented in Fig. 4. The obtained results show that the differences between the values of the control signal produced in FPGA and those in Simulink (Fig. 5) are smaller than the resolution of the output digital-to-analog converter. It proves that the considered procedure and code generator software correctly transformed the control system from the Simulink scheme. The presented tool enables fast, error free FPGA implementation of control algorithms specified on a high level of abstraction.

Keywords: controller implementation, controller architectures, programmable devices, FPGA.

1. Wstęp

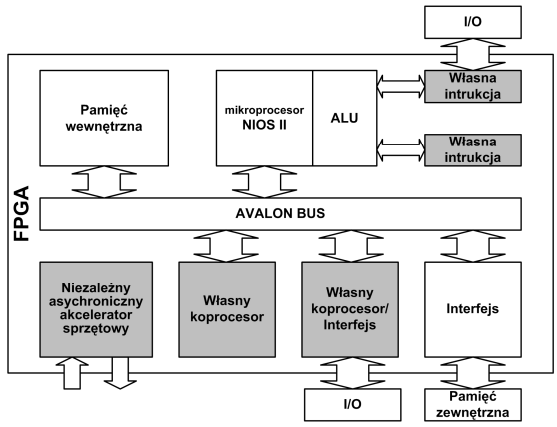
W przypadku złożonych algorytmów sterowania nie zawsze korzystne jest implementowanie sprzętowe całego algorytmu w FPGA, a zwłaszcza tych jego fragmentów, które mają nieregularną strukturę, lub są trudne do transformacji do postaci stałoprzecinkowej. Najlepszym rozwiązaniem w takich przypadkach wydaje się być realizacja mieszana, sprzętowo-programowa [1]. Stała się ona możliwa w związku z tym, że nowoczesne układy FPGA pozwalają zawrzeć w sobie cały system z mikroprocesorem (-ami), pamięcią, cyfrowymi układami peryferyjnymi i własnymi blokami przetwarzania sygnałów.

System w układzie (*System-on-Chip* - SoC) jest pojedynczym układem scalonym zawierającym w sobie cały system: mikroprocesor(y), koprocesory, jednostki przetwarzania sygnałów, układy peryferyjne, pamięci, interfejsy komunikacyjne itd. Implementacja SoC w układach FPGA, nazywanych wtedy też *System-on-a-Programmable-Chip* (SoPC) charakteryzuje się kilkoma zaletami. Umożliwia eksperymentowanie z charakterystykami wydajności, nawet w późnych fazach procesu projektowania. Wydłuża cykl życia produktu przez możliwość aktualizowania wersji sprzętu i oprogramowania lub dostosowywania funkcjonalności do specyficznych potrzeb konkretnego użytkownika lub zastosowania, bez potrzeby modyfikacji płyty drukowanej (fizycznej warstwy sterownika).

Dostępne możliwości zostaną bliżej wyjaśnione na przykładzie SoPC opartego na układzie FPGA z rodziny Stratix firmy Altera i wprogramowanego (*soft-core processor* – niewykonanego „na sztywno" w krzemie) 32-bitowego mikroprocesora Nios II typu RISC.

Typową architekturę przedstawiono na rys. 1 System jest zbudowany ze standardowych komponentów (białe tło) oraz z komponentów stworzonych przez projektanta (szare tło) połączonych magistralą Avalon, zbudowaną ze specjalizowanych zasobów układu. W systemie może współistnieć wiele mikroprocesorów Nios II pracujących na wspólnej magistrali lub niezależnie. Architektura Niosa II umożliwia dodawanie własnych instrukcji, definiowanych przez użytkownika. Instrukcje te są jedną z metod na zwiększenie osiągów systemu przez rozszerzenie jednostki arytmetyczno-logicznej (ALU) procesora o własne operacje realizowane sprzętowo.

Część programowa, jak i akceleratory sprzętowe powinny być projektowane i implementowane w sposób zgodny z metodologią projektowania mechatronicznego [2] i pozwalającymi na symulację części algorytmu realizowanych programowo oraz sprzętowo razem z pozostałą częścią systemu mechatronicznego w procesie wirtualnego prototypowania. Niestety nie istnieją narzędzia pozwalające na pełną realizację tego postulatu [3].



Rys. 1. Typowa architektura systemu w programowalnym układzie scalonym (SoPC), opartego na układzie FPGA firmy Altera i wprogramowanym mikroprocesorze Nios II; bloki z szarym tłem przedstawiają niestandardowe, własne komponenty projektanta

Fig. 1. Typical architecture of system on a programmable chip (SoPC), based on Altera FPGA and Nios II soft-processor; gray blocks are custom designed components

2. Założenia opracowanej metody

Głównymi celami podczas tworzenia procedury implementacji algorytmów przetwarzania sygnałów w układach FPGA było skrócenie czasu realizacji tej fazy projektu i zmniejszenie jej kosztów poprzez:

1) ograniczenie liczby i zakresu dokonywanych ręcznie transformacji algorytmu,
2) zastosowanie, tam gdzie to możliwe, komercyjnie dostępnego oprogramowania,
3) utrzymanie fizycznej odrębności kodu nienależącego bezpośrednio do implementowanego algorytmu,
4) ścisłą integrację z Simulinkiem,
5) umożliwienie ponownego wykorzystywania istniejącego, sprawdzonego kodu HDL,
6) jak największą niezależność od konkretnej technologii (producenta półprzewodników),
7) możliwość symulacji w Simulinku tej formy zapisu algorytmu, która jest bezpośrednio automatycznie syntezowana.

W pierwszej kolejności, w oparciu o dostępne komercyjnie narzędzia, została opracowana procedura implementacji algorytmów przetwarzania sygnałów w układach FPGA, spójna z ogólnymi metodami i technikami projektowania mechatronicznego. Następnie pewne kroki tej procedury, zwłaszcza te, wymagające podatnego na błędy przekształcania opisu algorytmu (kodowania) zostały zautomatyzowane, poprzez stworzenie generatora kodu ze schematu Simulinka.

3. Opracowana procedura implementacji

Początkowa specyfikacja algorytmu w postaci schematu Simulinka wymaga kilku przekształceń [4]. Pierwszym z nich jest dyskretyzacja w czasie, a drugim dyskretyzacja amplitudy (kwantyzacja). Zastosowanie wyłącznie arytmetyki stałoprzecinkowej pozwala na znaczącą redukcję kosztu realizacji algorytmu i czasu jego wykonywania. Tym niemniej należy zachować ostrożność, ponieważ reprezentacja stałoprzecinkowa z trudem radzi sobie z sygnałami o dużej dynamice, których amplitudy zmieniają się w zakresie kilku lub więcej rzędów wielkości. Należy ponadto zwrócić uwagę na wpływ efektów charakterystycznych dla obliczeń stałoprzecinkowych, takich jak przepełnienie i nasycenie. Dlatego po dokonaniu powyższych transformacji działanie algorytmu powinno zostać sprawdzone poprzez symulację. W efekcie tych działań algorytm dzielony jest na dwie części: stało- i zmiennoprzecinkową. Część zmiennoprzecinkowa zostanie zaimplementowana programowo, a część stałoprzecinkowa może zostać zaimplementowana sprzętowo (krytyczne fragmenty) albo programowo (pozostałe fragmenty). W tym miejscu procedury należy wyodrębnić obliczenia niezwiązane bezpośrednio z algorytmem przetwarzania sygnałów, które mogą, lub powinny, np. ze względu na wymaganie odporności na awarie oprogramowania, być wykonywane niezależnie i które zostaną zaimplementowane jako komponenty sprzętowe, działające niezależnie od magistrali Avalon. W przypadku sterownika dla robota może to być system zabezpieczeń.

Następnie, w czwartym kroku, algorytm należy zakodować w języku C/C++, jako tej formie, która będzie później (krok szósty i siódmy) transformowana dalej już całkowicie automatycznie. W przypadku fragmentów przeznaczonych do implementacji programowej kod ten zostanie skompilowany przez kompilator dla mikroprocesora Nios II. W przypadku fragmentów przeznaczonych do implementacji sprzętowej, kod zostanie przetworzony przez odpowiednie oprogramowanie do języka VHDL i skompilowany do plików konfigurujących układ FPGA. Kod w języku C/C++ posiada jeszcze trzy zalety. Po pierwsze może, poprzez dodanie odpowiednich interfejsów, zostać jądrem s-funkcji Simulinka, co pozwala symulować zakodowane w C/C++ fragmenty algorytmu tak, jak inne bloki (transformację do języka C/C++ można przeprowadzać etapami i łatwo weryfikować jej poprawność). Po drugie, pewne operacje, takie jak manipulacje na bitach, interfejsy urządzeń zewnętrznych i skalowanie danych dużo prościej jest zapisać od razu w języku C/C++ niż składać ze standardowych bloków Simulinka. Po trzecie wreszcie, symulacja kodu pozwala na określenie krytycznych fragmentów algorytmu, czyli tych, które wymagają najdłuższego czasu obliczeń i powinny być zrealizowane sprzętowo [5]. Jeżeli krytyczne fragmenty występują w kodzie zmiennoprzecinkowym, to należy przeprowadzić analizę najczęściej występujących w nich operacji i przeznaczyć je do realizacji sprzętowej za pomocą własnych instrukcji mikroprocesora Nios II. W piątym kroku procedury kod C/C++ dla sprzętowych operacji stałoprzecinkowych o niestandardowej długości słowa i precyzji jest tworzony z użyciem AR|T Library. Kod ten musi być zgodny ze specyfikacją magistrali Avalon, aby umożliwić połączenie akceleratorów sprzętowych z magistralą i wymianę danych po tej magistrali między częścią sprzętową i programową. W tym celu uzupełniany jest o opis mechanizmów sprzętowych, wymaganych przez magistralę: zestaw rejestrów wejściowo-wyjściowych oraz obsługę sygnałów sterujących magistrali Avalon. Dla umożliwienia uniwersalnego i zgodnego z przyjętymi standardami korzystania z komponentów sprzętowych w części algorytmu realizowanej sprzętowo, należy utworzyć także ich sterowniki, będące elementem warstwy abstrakcji sprzętowej (*Hardware Abstraction Layer – HAL*), odseparowującej konkretną architekturę systemu mikroprocesorowego od oprogramowania użytkowego. Sterowniki, oprócz wymiany danych między częścią programową i sprzętową algorytmu oraz ich synchronizacji, mają za zadanie właściwe przygotowanie danych wyjściowych oraz właściwą interpretację danych wejściowych. Kod C/C++ części zmiennoprzecinkowej zawiera wywołania funkcji obsługujących akceleratory sprzętowe (sterowników HAL) i instrukcje własne mikroprocesora. W celu uniknięcia tworzenia kilku wersji tego samego kodu, ciała funkcji realizujących właściwy, implementowany algorytm zapisywane są w osobnych plikach. Pliki te są następnie, w zależności od etapu procedury, dołączane do plików zawierających: opis interfejsów magistrali Avalon (podczas syntezy plików do programowania układu FPGA), deklaracje funkcji sterowników HAL (podczas kompilacji części programowej na mikroprocesor Nios II), bądź interfejsy s-funkcji (podczas kompilacji dla symulacji w Simulinku). W ten sposób podczas symulacji używany jest ten sam kod, który później służy do właściwej implementacji, co daje pewność zgodności sposobu działania zakodowanego algorytmu podczas symulacji z działaniem podczas eksperymentu, gdy jest on realizowany już w FPGA.

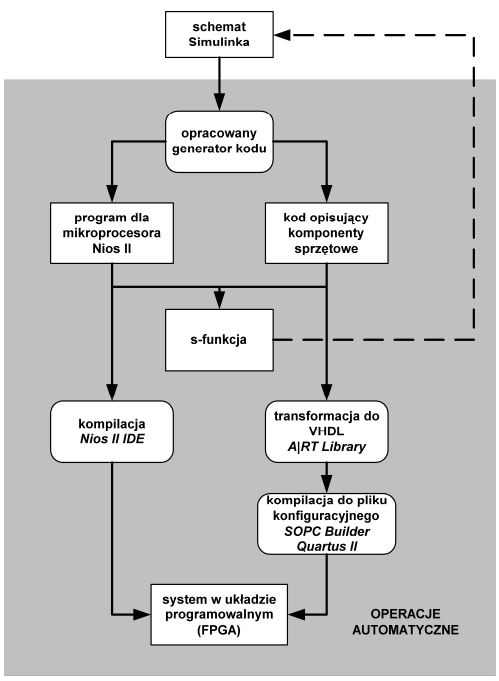
Na tym etapie procedury są już podjęte decyzje, które fragmenty algorytmu będą implementowane programowo (zakodowane w ANSI C), a które sprzętowo (zakodowane w C++ z użyciem AR|T Library).

Następnie, w kroku szóstym, należy skompilować kod przeznaczony do implementacji sprzętowej do języka VHDL za pomocą programu AR|T Builder. Końcowym, siódmym etapem, jest połączenie własnych elementów sprzętowych, w języku VHDL z pozostałą częścią systemu (m.in. mikroprocesorem) w programie SOPC Builder, w wyniku czego otrzymuje się kompletny opis warstwy sprzętowej w języku VHDL oraz pliki konfiguracyjne dla kompilatora dla mikroprocesora Nios II. Na podstawie tego kodu VHDL, program Quartus II generuje plik do zaprogramowania układu FPGA. W zaprogramowanym układzie można uruchamiać kod C opisujący część algorytmu realizowaną programowo, skompilowany kompilatorem GNU, a środowisko uruchomieniowe Eclipse IDE pozwala na kontrolę i uruchamianie programu w czasie rzeczywistym.

4. Automatyczna generacja kodu

Przedstawiona w rozdziale 3 procedura implementacji algorytmów przetwarzania sygnałów w systemie w układzie programowalnym uwalnia projektanta od najbardziej pracochłonnej i podatnej na błędy czynności, czyli kodowania w języku opisu sprzętu, ale w dalszym ciągu wykazuje pewne niedogodności. Najważniejszymi z nich są:

1) pracochłonność i podatność na błędy związane z czynnikiem ludzkim tj. kodowania zachowania bloków Simulinka w języku C/C++, w tym z użyciem biblioteki stałoprzecinkowej A|RT Library,
2) konieczność posiadania przez projektanta specjalistycznej wiedzy na temat działania magistrali Avalon oraz tworzenia interfejsów sprzętowych i programowych (sterowników HAL) dla poszczególnych elementów systemu mikroprocesorowego.

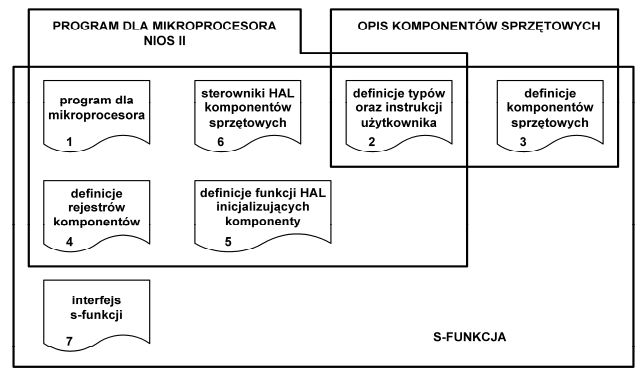


Rys. 2. Transformacje opisu implementowanego algorytmu w przypadku wykorzystania opracowanego generatora
Fig. 2. Transformations of the implemented algorithm description when using the developed generator

W celu wyeliminowania tych dwóch problemów zostało opracowane oprogramowanie automatycznie generujące ze schematu Simulinka kod w języku C/C++ opisujący program dla mikroprocesora oraz komponenty (akceleratory) sprzętowe współpracujące z tym mikroprocesorem [6]. Struktura fragmentów kodu, realizujących właściwy algorytm jest uniwersalna i przenoszalna, natomiast fragmenty realizujące współpracę części sprzętowej i programowej oraz integrującej część sprzętową z magistralą systemową są dedykowane dla układów FPGA firmy Altera i mikroprocesora Nios II. Przy wykorzystaniu opracowanego generatora kodu zadanie projektanta sprowadza się do spartycjonowania algorytmu na schemacie Simulinka na części realizowane programowo i sprzętowo, określenia wymaganych parametrów części sprzętowej i weryfikacji wygenerowanego kodu poprzez symulacje (rys. 2).

Generator działa dwuprzebiegowo: najpierw analizowany jest plik schematu Simulinka i tworzony na jego podstawie plik z uporządkowanym opisem topologii i funkcjonalności schematu z pominięciem informacji o graficznej reprezentacji modelu, a w drugim etapie następuje właściwa generacja kodu C/C++. Zastosowanie pośredniego opisu schematu pozwala na uniezależnienie operacji generowania kodu od zmiennego, w zależności od wersji, opisu schematu przez Simulink. Efektem działania generatora jest zbiór plików (rys. 3) umożliwiających:

1) syntezę komponentów sprzętowych, realizujących części algorytmu przeznaczone do implementacji sprzętowej,
2) kompilację programu dla mikroprocesora Nios II, realizującego części algorytmu przeznaczone do implementacji programowej,
3) kompilację s-funkcji pozwalającej na symulację w Simulinku działania zaimplementowanego algorytmu z dokładnością do pojedynczego bitu i cyklu zegara, poprzez wykorzystanie bez zmian ciał funkcji używanych w 1) i 2).



Rys. 3. Pliki tworzone przez generator
Fig. 3. Files created by the generator

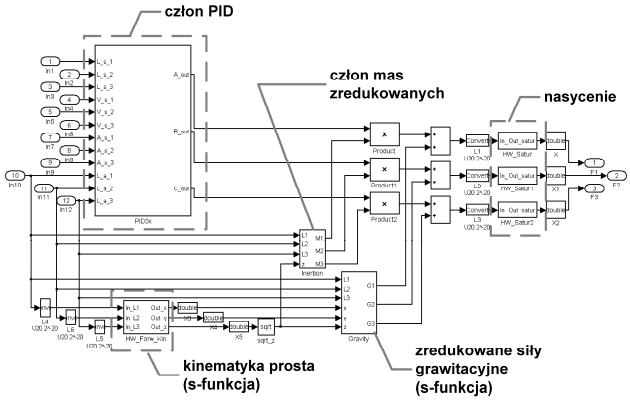
Wygenerowane pliki z kodem C/C++ opisują implementację algorytmu przetwarzania sygnałów zawartego w schemacie Simulinka, będącym punktem wyjścia opracowanej procedury. Aby stworzyć kompletny system w układzie programowalnym (SoPC) należy uzupełnić ten kod o elementy sprzętowe i programowe związane z topologią konkretnego obwodu drukowanego, w którym znajduje się układ FPGA. Ich zadaniem jest obsługa układów peryferyjnych, takich jak zewnętrzne pamięci, przetworniki analogowo-cyfrowe i cyfrowo-analogowe, interfejsy czujników i stopni mocy, klawiatur, wyświetlaczy, zegarów, przerwań, ewentualnych interfejsów do komunikacji z innymi urządzeniami itp. Przy tym te części systemu mikroprocesorowego, które są zależne od konkretnego obwodu drukowanego, są tworzone raz i nie zmieniają się przy zmianie implementowanego algorytmu. Natomiast części systemu realizujące implementowany algorytm przetwarzania sygnałów są generowane automatycznie. Dlatego należy uznać opracowany sposób postępowania, przy użyciu stworzonych narzędzi (generatora kodu) za szybkie prototypowanie algorytmów przetwarzania sygnałów na sprzęcie docelowym.

5. Implementacja algorytmu sterowania robotem równoległym

Działanie generatora kodu zostało sprawdzone przy implementacji algorytmu sterowania robotem równoległym [7]. Ze względu na krótki okres próbkowania i wysokie wymagania obliczeniowe algorytmu sterowania, jako docelową platformę sprzętową sterownika wybrano programowalny układ cyfrowy FPGA z rodziny Stratix firmy Altera z wprogramowanym mikroprocesorem

Nios II. Nadrzędnym celem przeprowadzenia tego fragmentu projektu było przetestowanie opracowanego generatora kodu i sprawdzenie poprawności implementacji, dlatego rozmyślnie wybrano taką formę prawa sterowania z modelem strukturalnym [7], która pozwoliłaby jak najlepiej sprawdzić działanie generatora dla różnego rodzaju bloków Simulinka, z niższym priorytetem traktując kwestię osiągnięcia wysokiej wydajności końcowej postaci algorytmu.

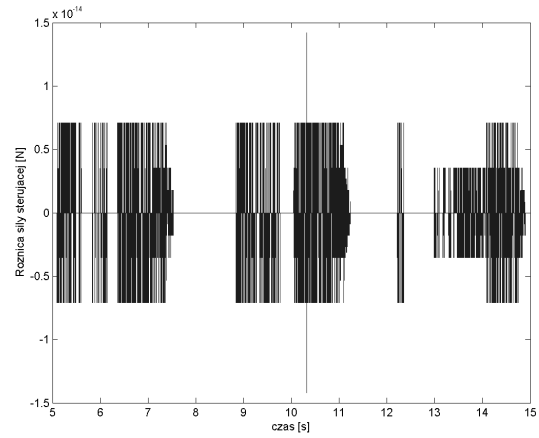
Schemat Simulinka, który stanowił źródło dla generowanego kodu, przedstawia rys. 4. Człon PID oraz mas zredukowanych z modelu dynamiki odwrotnej manipulatora zostały złożone ze standardowych bloków Simulinka, natomiast równania rozwiązania zadania prostego kinematyki oraz członu zredukowanych sił grawitacyjnych zostały zrealizowane jako s-funkcje. Nasycenie wartości sygnału sterującego na wyjściu odzwierciedla ograniczenia fizyczne napędów. Człon PID, równania kinematyki oraz nasycenia zostały przeznaczone do realizacji sprzętowej, jako komponenty własne (akceleratory sprzętowe) systemu mikroprocesorowego, pozostałe części natomiast do realizacji programowej wspomaganej instrukcjami własnymi mikroprocesora Nios II dla dodawania, mnożenia, obliczania pierwiastka kwadratowego i odwrotności liczb zmiennoprzecinkowych pojedynczej precyzji. W ten sposób w części sprzętowej i programowej, znajdują się fragmenty tak w formie połączeń standardowych bloków jak i s-funkcji.



Rys. 4. Schemat Simulinka, z którego był automatycznie generowany kod sterownika; szarymi przerywanymi liniami zaznaczone są fragmenty przeznaczone do realizacji sprzętowej
Fig. 4. Simulink scheme used for automatic generation of a controller code; fragments inside grey dashed lines are to be implemented in hardware

Wygenerowany kod był dołączany do projektu zawierającego komponenty sprzętowe i programowe wymagane do obsługi urządzeń peryferyjnych karty z układem FPGA, takich jak przetworniki cyfrowo-analogowe, interfejsy enkoderów, pamięci zewnętrzne, generator sygnałów zegarowych, interfejs magistrali PCI, oraz niezbędnych komponentów wewnętrznych systemu w układzie programowalnym, takich jak kontroler przerwań i układ licznikowo-czasowy, które są wykorzystywane m.in. do odmierzania okresu próbkowania i cyklicznego wykonywania automatycznie wygenerowanego kodu. Ta część projektu, która jest dołączana do generowanego kodu pozostaje stała i można jej używać bez zmian, pod warunkiem zachowania takiego samego okresu próbkowania, do różnych wersji algorytmu sterowania, których kod jest generowany automatycznie, realizując w ten sposób szybkie prototypowanie na sprzęcie docelowym.

Aby uczynić porównanie wyników działania bardziej wiarygodnym, do weryfikacji na zaprogramowanym układzie FPGA użyto sygnałów zarejestrowanych w trakcie eksperymentu przeprowadzonego na etapie szybkiego prototypowania. Dzięki temu istnieje pewność, że obydwie realizacje algorytmu: na sprzęcie do szybkiego prototypowania i zaimplementowana w FPGA działały na dokładnie tych samych sygnałach i ewentualne różnice wyników są skutkiem błędów lub niedokładności implementacji. Testy wykazały zgodność wyników lepszą niż 10^{-13} N (rys. 5). Jest ona wyższa niż rozdzielczość przetworników cyfrowo-analogowych, których wyjścia są podawane na stopnie mocy silników. Zgodność wyników potwierdza to, że opracowana metoda transformacji opisu systemu ze schematu blokowego Simulinka nie wprowadza zniekształceń w działaniu implementowanego algorytmu, oraz że wszystkie komponenty dozwolone przy tworzeniu systemu w oparciu o generator kodu działają zgodnie z założeniami.



Rys. 5. Różnica bezwzględna sterowania (siła w napędzie 1) wyznaczonego w FPGA i Simulinku
Fig. 5. Absolute difference between the control signal (force in the drive 1) calculated in FPGA and in Simulink

6. Podsumowanie weryfikacji opracowanego sterownika

Przeprowadzona weryfikacja działania opracowanego generatora kodu dowiodła, że działa on poprawnie. Wartości sił sterujących poszczególnymi osiami robota są jednakowe dla obydwu przypadków: obliczeń w Simulinku i w FPGA. Oznacza to, że testowany generator kodu nie wprowadza zmian do transformowanego układu i uzyskuje się pełną zgodność systemu stworzonego w Simulinku i systemu w układzie reprogramowalnym oraz, że wszystkie komponenty dozwolone przy tworzeniu systemu w oparciu o prezentowaną metodę działają zgodnie z założeniami. Daje to możliwość testowania projektowanego systemu na wczesnym etapie jego powstawania.

Praca naukowa finansowana ze środków na naukę w latach 2007-2010 jako projekt badawczy N501 030 32/2508

7. Literatura

[1] Edwards M.D., Forrest J., Whelan A.E.: Acceleration of software algorithms using hard-ware/software co-design techniques. J. of Systems and Architecture, vol. 42, 1996/97, pp. 697-707.
[2] Amerongen van J.: Mechatrionic design. Mechatronics vol. 13, pp 1045-1066.
[3] T. Kambe, A.Yamada and M. Yamaguchi, Trend of system level design and approach to C-based design, Microelectronics Journal, Vol. 33, 2002 pp. 875-880.
[4] Petko M., Karpiel G., Uhl T.: Implementacja algorytmów sterowania w układach FPGA na przykładzie robota równoległego. PAK nr 5/2006, s. 27-30.
[5] Petko M., Karpiel G.: Hardware/Software Co-design of Control Algorithms. Proceedings of the 2006 IEEE International Conference on Mechatronics and Automation, 2006, pp. 2156-2161, ISBN: 1-4244-0466-5.
[6] Lubieniecki M., Staworko M.: Automatyczna implementacja programowo-sprzętowa algorytmów sterowania w układach FPGA. Praca dyplomowa magisterska wykonana pod opieką dr inż. M. Petko. AGH, 2007.
[7] Petko M.: Sterowanie neuronowe robotem równoległym – projekt i implementacja PAK, nr 5/2006, 2006 s. 31-34.

Artykuł recenzowany